الگویی که قصد توضیح آن را داریم الگوی Strategy می باشد. این الگو یک الگوی رفتاری یا Behavioral می باشد. الگوهای رفتاری با الگوریتم ها و ارتباط بین آن ها درگیر می باشند. اگر برنامه ما یک الگوریتم یکتا را پیاده سازی می کند گرچه ممکن است از چندین کلاس استفاده کند ولی به خاطر بالا رفتن پیچیدگی ممکن است که برای مدیریت برنامه و نگهداری کد مشکلاتی به جود بیاید. ما با استفاده از الگوهای طراحی رفتاری سعی می کنیم که از این پیچیدگی ها کم کنیم. حال به بررسی الگوی طراحی Strategy می پردازیم.



الگوی طراحی Strategy

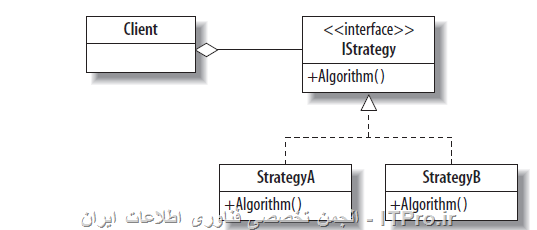
این الگو به این منظور استفاده می شود که یک الگوریتم را از کلاس میزبان جدا کرده و در کلاس دیگر نگهداری نماید. ممکن است که ما در برنامه خود چندین الگوریتم(استراتژی) داشته باشیم. حال اگر بخواهیم همه این الگوریتم ها را در برنامه خود داشته باشیم در صورت عادی باید از عبارات شرطی بسیاری استفاده کنید و همین عمل باعث می شود که پیچیدگی برنامه به شدت بالا برود. الگوی strategy کاربر را قادر می سازد که از داخل یک گروه از الگوریتم ها یک الگوریتم را برای استفاده انتخاب نماید.

شرح الگوریتم

اگر بخواهیم یک مثال ساده از الگوی Strategy بزنیم می توانیم برنامه ای را مثال بزنیم که با استفاده از روش های متفاوت مرتب سازی یک سری اعداد را مرتب سازی می کند. الگوریتم های متفاوتی برای مرتب سازی وجود دارد که می توان به QuickSort و یا MergeSort اشاره کرد. هرکدام از این روش ها می توانند در شرایط خاصی بهترین انتخاب موجود باشند. می توان تصمیم گیری این شرایط را با استفاده از الگوی Strategy پیاده سازی نمود. به این شکل که یک مجموعه از ورودی ها را داریم. ولی هرکدام از روش های مرتب سازی را به صورت جداگانه پیاده سازی می کنیم. ولی ورودی ها و خروجی ها ی یکسانی داشته باشند. در این صورت با توجه به شرایط ورودی ها را به یک الگوریتم داده و خروجی آن را دریافت خواهیم کرد.

طراحی الگو

طراحی الگوی Strategy در قالب نمودار UML در شکل زیر نشان داده شده است. همانگونه که در شکل دیده می شود می توان از داخل استراتژی های موجود در یک گروه استراتژی مناسب انتخاب کرد.



کلاس هایی که در این الگو هستند به شرح زیر است: 

* Context: کلاسی که اطلاعات ساختاری برای اشیا IStrategy را در خود نگه می دارد. همان قسمتی از برنامه است که نوع استراتژی را انتخاب می کند.
* IStrategy: یک اینترفیس مشترک بین همه ی استراتژی ها تعریف می کند.
* StartegyA,StrategyB کلاس هایی که شامل الگوریتم های پیاده شده از اینترفیس IStrategy می باشد.

کلاس Context یک شی از نوع استراتژی انتخاب شده می سازد. کلاس Strategy ممکن است که به وضعیت Context دسترسی داشته باشد.

پیاده سازی

پیاده سازی برای الگوی Strategy نیازمند ویژگی های خاص و جدید C# نیست و می توان آن را به سادگی پیاده سازی کرد. در پیاده سازی کلاس Context یک نمونه از اینترفیس IStrategy ساخته می شود. کلاس استفاده کننده متد الگوریتم را از داخل Context فراخوانی می کند و این کلاس در خواست را به استراتژی مورد نظر هدایت می کند. در مثال زیر هر کدام از استراتژی ها یک متد move دارند و یک تولید کننده عدد تصادفی در کلاس استفاده کننده مشخص می کند که چه زمانی زمان تعویض الگوریتم می باشد. این مثال در کد زیر آورده شده است. 

using System;

// The Context

class Context {

// Context state

public const int start = 5;

public int Counter = 5;

// Strategy aggregation

IStrategy strategy = new Strategy1( );

// Algorithm invokes a strategy method

public int Algorithm( ) {

return strategy.Move(this);

}

// Changing strategies

public void SwitchStrategy( ) {

if (strategy is Strategy1)

strategy = new Strategy2( );

else

strategy = new Strategy1( );

}

}

// Strategy interface

interface IStrategy {

int Move (Context c);

}

/ Strategy 1

class Strategy1 : IStrategy {

public int Move (Context c) {

return ++c.Counter;

}

}

// Strategy 2

class Strategy2 : IStrategy {

public int Move (Context c) {

return --c.Counter ;

}

}

// Client

static class Program {

static void Main ( ) {

Context context = new Context( );

context.SwitchStrategy( );

Random r = new Random(37);

for (int i=Context.start; i<=Context.start+15; i++) {

if (r.Next(3) == 2) {

Console.Write("|| ");

context.SwitchStrategy( );

}

Console.Write(context.Algorithm( ) +" ");

}

Console.WriteLine( );

}

}

خروجی کد بالا به شکل زیر خواهد بود: 

/\* Output

65 4 || 5 6 7 || 6 || 7 8 9 10 || 9 8 7 6 || 7 || 6 5

66 \*/

برخی از نکات کلیدی درباره ی پیاده سازی الگوی Strategy عبارتند از:   
• اغلب کلاس Context شامل عبارت switching یا تعویض الگوریتم یا چند دستور شرطی تو در تو می باشد تا در شرایط متفاوت تصمیماتی گرفته شود   
• اگر استراتژی ها ساده هستند می توانند به شکل delegate پیاده سازی شوند و نیازی به ساختن کلاس نیست.   
• اگر بخواهیم برنامه را بعدا گسترش دهیم و الگوریتم های جدیدی توسط کاربران به برنامه اضافه کنیم می توانیم از Extension method ها استفاده کنیم.

استفاده از الگو

یکی از موارد استفاده از الگوی Strategy استفاده از این الگو در پیاده سازی تکنولوژی Linq در زبان C# می باشد که می توان به راحتی برای انواع پایگاه های داده و انواع مجموعه داده ها از آن استفاده کرد.   
به طور کلی در زمان های زیر از الگوی Strategy استفاده می کنیم. 

1. زمانی که کلاس های مرتبط زیادی داشته باشیم و این کلاس ها فقط در رفتارشان متفاوت باشند.
2. زمانی که برای انجام کاری الگوریتم های متفاوتی موجود باشد و بتوان شرایط انتخاب الگوریتم را توسط کدنویسی مشخص کرد.
3. زمانی که الگوریتم از داده هایی استفاده می کند که استفاده کننده ها نباید به آن کد دسترسی داشته باشند.

زمانی پیش می آید که Context نمی خواهد از هیچکدام از استراتژی های موجود استفاده کند در این صورت می گوییم که Context از استراتژی do nothing استفاده می کند. یعنی هیچ کاری انجام نمی دهد.